Data Wrangling with MongoDB

Alex Spanos

OpenStreetMap project: Hampshire, England, United Kingdom

1. Map used

I initially started to work on OSM data from Athens, Greece (by Metro Extracts), my hometown. After exploring the data however I noticed that language inconsistencies were present. As I feared that trying to address these inconsistencies would add a lot of time to the turnaround of the project, I opted to use an extract of Hampshire, England, United Kingdom where I currently live ("hampshire-latest.osm" by Geofabrik). The unzipped dataset size was 427MB - well within the minimum specifications in the rubric.

2. Problems encountered

During the initial exploration of the data prior to exporting to MongoDB I observed (as expected) various inconsistencies and issues, given the fact that they are human-entered.

For example, address information was not entered in a consistent manner in the "*address.street*" field. Based on the regular expression methodology shown in Lesson 6, I identified a multitude of values with case issues (fully capitalised, fully lower case etc) and generic problematic character issues. Additionally, after auditing this field based on regular expressions and returning "non-expected" values, (I updated my initial expectation after reviewing the first results) I noted other data entry issues in "*address.street*", such as:

- Typos (e.g. "Raod"): fixed programmatically using a mapping dictionary
- Postcodes: set to '-' programmatically using a regex for UK postcodes
- Housenumbers: not fixed, but could easily omit using regex
- Districts: few cases, not necessary to programmatically fix
- Few entries were descriptions rather than actual names (i.e. Junction of Drayton Lane Nr No. 159 Havant Road Drayton Portsmouth): fixed non-programmatically
- Few entries were plainly nonsensical (e.g. "X" or "S"): set to '-' non-programmatically

My overall initial impression, however, the entered data seemed fairly accurate based on my knowledge of the area; spellings were generally correct, postcodes belonged to the Hampshire area; I also recognised quite a few names of the "*amenity*" field values.

After loading the data into MongoDB however, I realised more shortcomings of the dataset in the various fields (e.g. highly inconsistent phone numbers); but most importantly the biggest shortcoming was missing data.

I used the following commands in order to interrogate the completeness of each top-level and lower-level field.

```
> element_list = ["amenity", "address", "address.city",
  "address.country", "address.place", "address.postcode",
  "address.street", "address.housenumber", "address.interpolation",
  "address.district", "address.site", "type", "id", "name",
  "node_refs", "pos", "phone", "created", "created.changeset",
  "created.timestamp", "created.uid", "created.user",
  "created.version", “cuisine”]
> MongoDB_element_dict = {}
> for entry in element_list:
     MongoDB_element_dict[entry]=[db.osm.find({entry:
     {"$exists":1}}).count(), round(100*db.osm.find({entry:
     {"$exists":1}}).count()/MongoDB_entries[0], 2)]
```

In the following Python dictionary, the first list element in the value lists represents how many times the field (represented by the dictionary key) occurs in the database and the second the relevant occurrence percentage.

Output 1

```
{'address': [32704, 1.45],
 'address.city': [19592, 0.87],
 'address.country': [17397, 0.77],
 'address.district': [7489, 0.33],
 'address.housenumber': [26718, 1.19],
 'address.interpolation': [237, 0.01],
 'address.place': [1362, 0.06],
 'address.postcode': [18978, 0.84],
 'address.site': [48, 0.0],
 'address.street': [29613, 1.31],
 'amenity': [15701, 0.7],
 'created': [2252685, 100.0],
 'created.changeset': [2252685, 100.0],
 'created.timestamp': [2252685, 100.0],
 'created.uid': [2252667, 100.0],
 'created.user': [2252667, 100.0],
 'created.version': [2252685, 100.0],
 'cuisine': [794, 0.04],
 'id': [2252685, 100.0],
 'name': [80957, 3.59],
 'node_refs': [261248, 11.6],
 'phone': [401, 0.02],
 'pos': [1991437, 88.4],
 'type': [2252685, 100.0]}
```

The “*address*” field is therefore present in 32704 out of the 2252685 entries in the database, corresponding to roughly 1.5%.

I also retrieved the completeness of fields nested within the "*address*" field; these are displayed in the following dictionary:

Output 2

```
{'address': 100.0,
 'address.city': 59.91,
 'address.country': 53.2,
 'address.district': 22.9,
 'address.housenumber': 81.7,
 'address.interpolation': 0.72,
 'address.place': 4.16,
 'address.postcode': 58.03,
 'address.site': 0.15,
 'address.street': 90.55}
```

The "*address*" field exists only if one of its sub-fields exists; from the above completion percentages it is evident that users tend to neglect its full completion, and usually fill out only "*housenumber*" and "*street*".

It is possible to programmatically fill out missing sub-fields based on information contained in the present ones. For example, when "*city*" is missing, it can be filled out with a high degree of certainty based on information in "postcode", "street", "district". This can be achieved with the use of a …… The same applies to "*district*". Additionally, "*country*" should always be set to "*GB*". There is a possibility to complete "*postcode*" and "*street*" based on existing information but that can most likely happen with a smaller degree of certainty.

Regarding the other fields, I was disappointed at the level of completeness of "*amenity*"; programmatically filling it out does not seem easy.

I was also curious regarding the level of completeness of "*cuisine*". Clearly the presence of this tag should imply the presence of amenity, so I performed the relevant sanity check using the following pymongo command:

```
> db.osm.find({"$and": [{"amenity": {"$exists": 0}}, {"cuisine":
  {"$exists": 1}}]}).count()
```

The command returned the number of documents where "*cuisine*" existed whilst "*amenity*" did not; the number was only 7 (out of 794 entries with amenity), which is good. The values for "*amenity*" for these can easily be fixed non-programmatically.

Subsequently I devised an aggregation pipeline to return the values occurring for "*amenity*" (when "*cuisine*" existed), counting their frequency and sorting them in descending order:

```
> pipe1 = {"$match": {"$and": [{"amenity": {"$exists": 1}},
  {"cuisine": {"$exists": 1}}]}}
> pipe2 = {"$group":{"_id": "$amenity", "count":{"$sum": 1}}}
> pipe3 = {"$sort": {"count":-1}}
> pipe4 = {"$project": {"name": "$amenity", "count":"$count"}}
> pipeline=[pipe1, pipe2, pipe3, pipe4]
> test = db.osm.aggregate(pipeline)
> for i in test:
        pprint(i)
```

This returned:

Output 3

```
{'_id': 'fast_food', 'count': 401}
{'_id': 'restaurant', 'count': 292}
{'_id': 'cafe', 'count': 90}
{'_id': 'pub', 'count': 4}
```

Some extra data cleaning is clearly required (underscore removal, caps).

Corresponding counts for the "*amenity*" values when "*cuisine*" did not exist was:

Output 4

```
{'_id': 'pub', 'count': 1088}
{'_id': 'cafe', 'count': 266}
{'_id': 'restaurant', 'count': 178}
{'_id': 'fast_food', 'count': 100}
```

Whilst it makes sense for the majority of pubs and cafes to not have an associated cuisine, it is unfortunate that this occurs for approximately 25% of fast food joints and 38% of restaurants. On the bright side, users have been very meticulous in their listing of pubs on OpenStreetMap.

Other observations here included that "*name*" was frequently missing when "*cuisine*" was missing; conversely when "*cuisine*" was present "*name*" was never missing.

3. Data overview

- File sizes:
  - Hampshire-latest.osm: 427MB
  - Hampshire-latest.osm.json: 469MB

- Number of documents:

  ```
  > db.osm.find().count()

  2252685
  ```

- Number of nodes and ways:

- Nodes: 1991437
- Ways: 261248

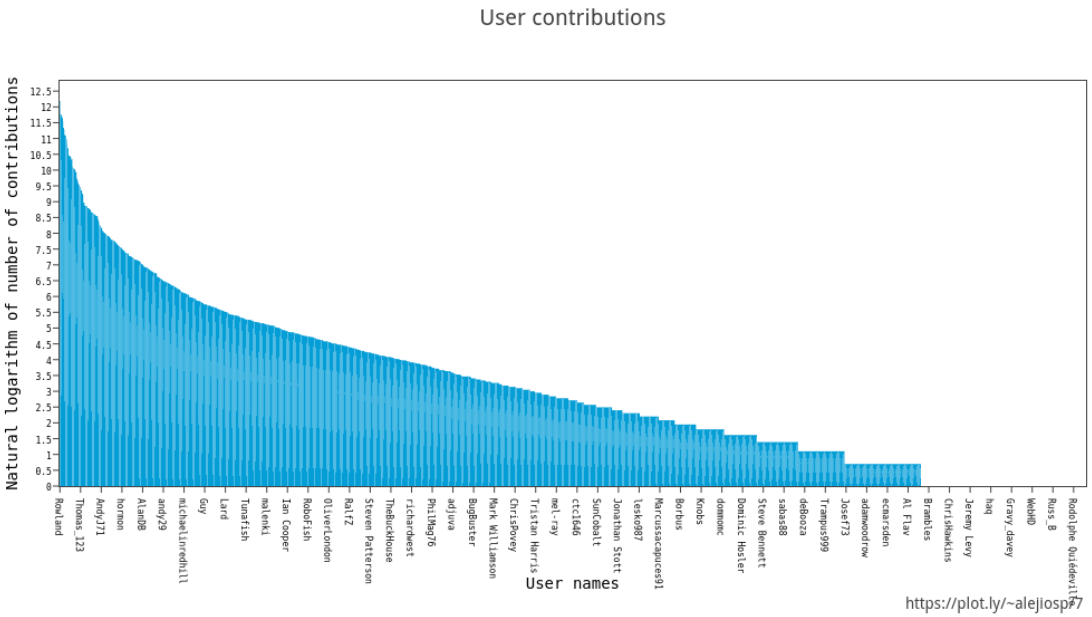
- Number of unique users:

```
> len(db.osm.distinct("created.user"))
```

1687

- User contribution distribution:
  In the bar chart of Figure 1, seen below, frequencies of the contributions of individual users can be visualised; contributions are in logarithmic scale.
  Clearly user contributions are highly skewed…

Figure 1



4. Additional ideas

OpenStreetMaps is currently the most significant non-corporate owned digital map of the world; therefore its value is undisputed. In my opinion, for OSM to be more competitive with Bing Maps and Google Maps, it needs to expand its user community and impose stricter acceptance criteria in data entry.

For example by automatically identifying wrongly entered data (such as entering the city name in the wrong field) and storing this in a flag variable would allow users to more easily audit and correct these. Automatic identification could be implemented based on a probabilistic algorithm that uses data from nearest-neighbours.

Additionally a gamification approach for smartphones would obviously be greatly beneficial.

5. Conclusions

MongoDB constituted a great platform for querying the OSM database. I was particularly impressed with the run time for my various pymongo commands.

Regarding data quality, whilst a multitude of issues are present in the dataset, the overall user effort is commendable. An improved systemisation of data entry may allow the development of a very strong product.